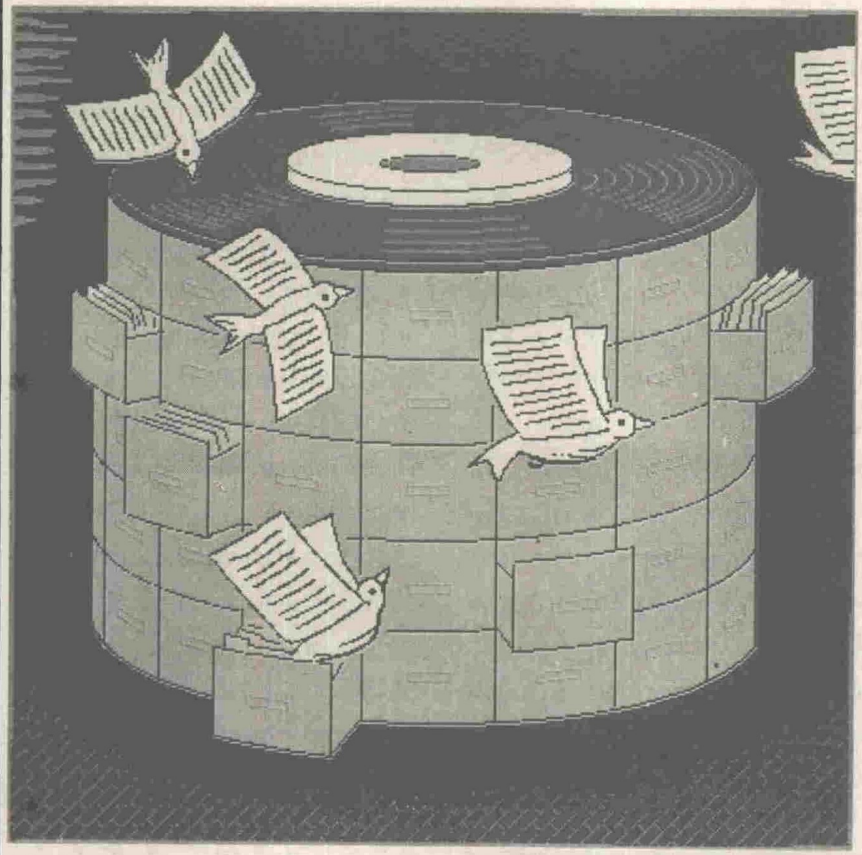
ดร.ปัญญา เปรมปรีดิ์

เทคนิคการออกแบบแฟ้มข้อมูล



ค้นหาทางเลือกที่ดีที่สุดในการออกแบบแฟ้มข้อมูลเพื่อช่วยแก้ปัญหาเมื่อไฟดับหรือหยุดโปรแกรมกะทันหัน...

ผู้ใช้เครื่องระดับไมโครคอมพิวเตอร์นั้นต้องเจอกับปัญหาไฟฟ้าดับและการหยุดโปรแกรมโดยกะทันหันอยู่เสมอ เมื่อเกิดเรื่องขึ้นแต่ละครั้งก็จะมีคำถามว่าแฟ้มข้อมูล (File) ที่อยู่ในระบบงานนั้นจะยังใช้การได้หรือไม่ บางท่านก็จะใช้วิธีเอาสำเนาล่าสุดมาก็อบปี้ทับลงไป แล้วดำเนินการอัพเดทซ้ำใหม่ไปเลย วิธีนั้นก็ดูจะง่ายดีแต่ถ้าท่านไม่ได้ทำสำเนาเอาไว้เลยก็คงจะต้องร้องไห้ หรือถ้าทำไว้นานแล้วก็ต้องไปค้นว่าทำไว้เมื่อไหร่ และจะต้องเอารายการอะไรมาอัพเดตย้อนหลังบ้าง เรื่องนี้จึงเป็นปัญหาพอสมควร จากปัญหาอันนี้เราจึงมีคำถามว่า เราจะออกแบบแฟ้มข้อมูลที่สามารถใช้การได้อยู่เสมอเมื่อเกิดเหตุการณ์ดังกล่าวได้หรือไม่ และทำกันอย่างไร เรื่องนี้ผมมีวิธีการที่อาจพอช่วยได้ แต่ก็มีเงื่อนไขที่จำเป็นต้องระมัดระวังอยู่เล็กน้อย เทคนิคนี้ผมใช้มานานแล้วและยังไม่เคยพบว่าผิดพลาด แต่ผมก็ไม่กล้าที่จะรับประกันว่าจะใช้การได้ 100% เต็ม สำหรับงานที่สำคัญมาก ๆ นั้นผมขอแนะนำให้ทำโปรแกรมเพื่อเช็คดูสภาพความสอดคล้องของแฟ้มข้อมูลเสียก่อน แต่จะใช้หรือไม่ใช้ก็อยู่ที่ท่านจะตัดสินใจ

มาทำความเข้าใจกับปัญหาเสียก่อน

ระบบควบคุมจานแม่เหล็ก (ทั้งชนิดดิสเกตต์และฮาร์ดดิสก์) ในสมัยใหม่นี้มีความฉลาดไม่น้อยทีเดียว กล่าวคือ เมื่อเกิดไฟฟ้าดับไปนั้นตัวควบคุมและตัวเครื่องจานแม่เหล็กจะทำงานต่อไป (ด้วยไฟฟ้าที่ประจุไว้ใน capacitor) จนกว่าจะเสร็จคำสั่งครั้งสุดท้าย หรือคือ ถ้ากำลังอ่านข้อมูลอยู่มันก็จะทำงานจนกระทั่งได้ข้อมูลจากตำแหน่งที่ต้องการออกไป และถ้ากำลังทำการบันทึกข้อมูลอยู่ มันก็จะทำงานบันทึกข้อมูลลงไปจนจบบล็อกดังกล่าว ด้วยเหตุนี้เราจึงอาจตั้งสมมติฐานไว้ได้เลยว่า

จะไม่มีการบันทึกข้อมูลครึ่ง ๆ กลาง ๆ ทิ้งเอาไว้ในแผ่นจานแม่เหล็ก เรื่องนี้ท่านจะสังเกตได้ คือหลังจากไฟฟ้าดับโดยกะทันหัน แฟ้มข้อมูลทุกแฟ้มจะยังอ่านออกเสมอ ขนาดของเรคอร์ดหรือบล็อกยังถูกต้องและการจัดเรียงบิตต่าง ๆ ก็ยังคงมีความถี่เท่าเดิม

แล้วปัญหามันอยู่ที่ไหน

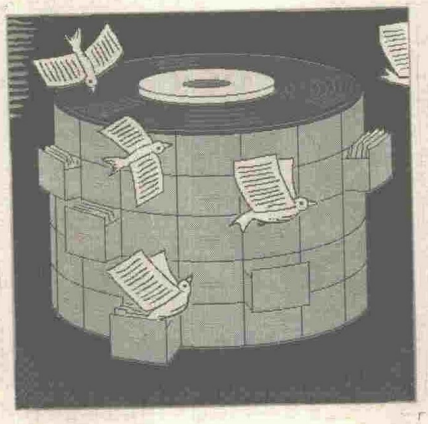
ปัญหาก็คือ แฟ้มข้อมูลที่เราใช้อยู่ในระบบงานต่าง ๆ นี้ บางทีเราสร้างข้อมูลเอาไว้หลายเรคอร์ด ซึ่งมีความสัมพันธ์กันเช่น บางเรคอร์ดเป็นดัชนี (Index) ซึ่งชี้ไปยังตำแหน่งที่อยู่ของข้อมูลจริง บางเรคอร์ดเก็บตัวเลขบอกจำนวนเรคอร์ดที่ได้ใช้ไปแล้ว การที่เรามีข้อมูลเก็บไว้หลายจุด แต่ต้องยันกันอยู่นี้ จะทำให้การอัพเดตหนึ่งครั้งต้องกระทบถึงข้อมูลหลายจุด ซึ่งเครื่องคอมพิวเตอร์จะทำงานเรียงไปตามลำดับที่เราสั่งเอาไว้ในโปรแกรมการทำงานเรียงตามลำดับนี้ ถึงแม้ว่าจะทำงานต่อเนื่องกันได้อย่างรวดเร็ว แต่ไฟฟ้าอาจจะดับในขั้นตอนใดขั้นตอนหนึ่งก็เป็นได้ และทำให้โปรแกรมหยุดระหว่างที่ยังอัพเดตข้อมูลได้ไม่สมบูรณ์ครบถ้วน นี่เป็นปัญหา เพราะเมื่อเรามาเริ่มเดินโปรแกรมกันใหม่ก็อาจพบว่าข้อมูลในเรคอร์ดต่าง ๆ ไม่สอดคล้องกัน เช่น ถ้าเราสั่งอัพเดตตัวดัชนีไปก่อนแล้วไฟฟ้ามาดับลงพอดี การอัพเดตตัวเรคอร์ดที่เก็บข้อมูลจริง ๆ ก็อาจยังไม่เกิดขึ้น เมื่อมาเดินโปรแกรมกันใหม่ก็อาจพบว่ามีตัวดัชนีเกินมา 1 ตัว และอาจมีผลทำให้โปรแกรมออกรายงานบางโปรแกรมไม่ทำงาน หรือทำงานผิดพลาดไปหมดก็ได้

นอกจากปัญหาในลักษณะดังกล่าวมาข้างต้นนี้แล้ว ระบบควบคุมหรือโอปะเรติ้ง ซิสเต็มส์ (Operating System) ของบางยี่ห้อนั้นก็ได้เข้ามายุ่งกับขั้นตอนการบันทึกและการอ่านข้อมูลของจานแม่เหล็กอยู่ด้วย ยกตัวอย่างเช่น มีการกักเก็บข้อมูลที่ควรจะส่งลงไปบันทึกในแผ่นจานแม่เหล็กเอาไว้ก่อน มีการจับเอาข้อมูลหลาย ๆ เรคอร์ดมาเชื่อมต่อกัน เพื่อให้เป็นบล็อกใหญ่ ๆ ฯลฯ สิ่งเหล่านี้ช่วยเพิ่มประสิทธิภาพในการทำงาน เช่น เรคอร์ดที่สั่งให้บันทึกลงแผ่นจานแล้วนั้น ตัวระบบควบคุมจะยังไม่ส่งออกไปบันทึกลงจานแม่เหล็กในทันที เพราะถ้ามีการสั่งให้อ่านกลับเข้ามาใหม่ก็จะไม่ต้องเสียเวลาอ่านจริง ตัวระบบควบคุมจะบอกได้เลยว่ามีข้อมูลนั้นอยู่แล้ว ส่วนการจับเอาเรคอร์ดหลาย ๆ เรคอร์ดมาวางต่อกันเป็นบล็อกใหญ่ ๆ นั้นจะประหยัดเนื้อที่ในจานแม่เหล็ก มันประหยัดได้เมื่อความยาวของเรคอร์ดของเราไม่ตรงหรือไม่ลงตัวกับขนาดของเซกเตอร์ในจานแม่เหล็ก กล่าวคือโดยปกติแล้วเครื่องจานแม่เหล็กจะอ่านหรือบันทึกได้ทีละเซกเตอร์ (บางคนเรียกว่าเซกเมนต์ ขนาดของเซกเตอร์จะคงที่ เช่น เซกเตอร์ละ 512 ไบต์ ถ้าเรคอร์ดของเรายาว 100 ไบต์ และไปบังคับให้บันทึกลงในทันทีที่สั่ง ตัวระบบควบคุมก็อาจต้องปล่อยเนื้อที่อีก 412 ไบต์ว่างเปล่า ซึ่งเป็นการสูญเสียที่สูงมาก ตัวระบบควบคุมที่ฉลาดพอก็จะพยายามจับเอาเรคอร์ดของเราจำนวน 5 เรคอร์ดมาเชื่อมต่อกันเป็น 1 บล็อก แล้วบันทึกลงแผ่นจานแม่เหล็กพร้อม ๆ กัน ด้วยวิธีนี้จะทำให้สูญเสียเพียง 12 ไบต์ต่อ 5 เรคอร์ด แต่วิธีนี้ทำให้ต้องอ่านขึ้นมาทั้งบล็อก เช่น ถ้าเราต้องการเรคอร์ดที่ 3 ตัวระบบควบคุมก็ต้องอ่านบล็อกแรกทั้งบล็อกขึ้นมา เวลาในการอ่านจะเท่า ๆ กับการกระจายไปเรคอร์ดละ 1 เซกเตอร์ มันจึงไม่มีข้อเสียหายอะไร เพียงแต่เราต้องมีเมมโมรีมากขึ้นอีกเล็กน้อย แถมมันยังจะช่วยลดเวลาในการทำงานลงได้ถ้ามีการสั่งให้อ่านข้อมูลที่เป็นเรคอร์ดที่อยู่ในบล็อกเดียวกันในเวลาต่อมา

เจ้าความชาญฉลาดของระบบควบคุม (OS) ดังกล่าวนี้สร้างปัญหาให้เราได้ เพราะเรคอร์ดที่ควรจะบันทึกลงไว้ในจานแม่เหล็กแล้วนั้นจะหายไปเมื่อไฟฟ้าเกิดดับลง เรื่องนี้เคยเป็นปัญหามากอยู่ในสมัยหนึ่ง เป็นมากจนทำให้ผู้สร้างภาษาบางภาษาต้องใส่คำสั่งสำหรับบังคับให้มีการบันทึกข้อมูลลงจานแม่เหล็กทันทีเมื่อเจอคำสั่งอย่างนั้น อย่างเช่น คำสั่ง FLUSH ในภาษาปาสคาล

นอกจากเรื่องของความชาญฉลาดของระบบควบคุมแล้ว เราก็ยังมีฮาร์ดแวร์และซอฟต์แวร์ที่เข้ามาต่อแยกระบบการบันทึกและอ่านข้อมูลอีกมาก เช่น ตัวควบคุมจานแม่เหล็กของบางยี่ห้อ

ที่มีเมมโมรีอยู่ในตัว เวลาสั่งให้อ่านข้อมูลจากเซกเตอร์ใดเซกเตอร์หนึ่งนั้นจะไม่ทำแค่คำสั่ง แต่จะอ่านเอาเซกเตอร์ถัด ๆ ไปขึ้นมาอีกจำนวนหนึ่ง บางยี่ห้ออ่านขึ้นมา 8-16 K แต่บางยี่ห้อเอาขึ้นมาถึง 64 K ซึ่งเรียกว่า Disk Cache Memory มันจะช่วยเราได้ถ้าคำสั่งให้อ่านครั้งต่อไปมาตกในเซกเตอร์ถัด ๆ ไปเหล่านี้ กรณีที่คำสั่งให้อ่านครั้งถัดไปมาตกอยู่ในกลุ่มนี้เราเรียกว่าเกิดการ "Hit" ซึ่งตัวควบคุมจานแม่เหล็กก็จะไม่ต้องไปเสียเวลาอ่านจากจานแม่เหล็ก เวลาจะลดจาก 20-30 มิลลิวินาทีเหลือเป็น 50-60 ไมโครวินาทีเท่านั้น การทำงานก็จะเร็วขึ้นมาก เวลาโปรแกรมสั่งบันทึกข้อมูลลงแผ่นจานแม่เหล็กก็จะมีผลเช่นเดียวกัน คือถ้าสั่งบันทึกลงในเซกเตอร์ที่ถูกอ่านขึ้นมาในดิสค์แคชเมมโมรีแล้วนั้นเจ้าตัวควบคุมจานแม่เหล็กจะไม่ทำการบันทึกจริง มันเพียงแต่นำข้อมูลจากในโปรแกรมมาใส่ทับในตำแหน่งที่ตรงกันภายในดิสค์แคชเมมโมรี ด้วยวิธีนี้มันจะลดทอนเวลาได้ 20-30 มิลลิวินาทีอีกเช่นเคย เรื่องนี้เป็นปัญหาได้หนักยิ่งกว่าเก่า เช่น สมมติว่าแฟ้มข้อมูลของเรามีขนาดแค่ 20-30 K สิ่งที่จะเกิดขึ้นก็คือ ข้อมูลตกจานแม่เหล็กทั้งแฟ้มจะถูกนำขึ้นมาอยู่ในดิสค์แคชเมมโมรีจนหมด (สมมติว่าดิสค์แคช = 32 K) การอัพเดตที่เราทำมาตลอดวันก็จะอยู่ในดิสค์แคชเมมโมรี ดังนั้นถ้าไฟฟ้าเกิดมาดับลง หรือเราไปบูตเครื่องโดยยังไม่ได้สั่งให้จบโปรแกรม ข้อมูลทั้งหมดที่ทำมาทั้งวันก็จะหายไป และเราจะบอกว่าข้อมูลในจานแม่เหล็กจะยังเหมือนตอนก่อนใช้งานก็ไม่ได้ เพราะโปรแกรมของเราอาจไม่ได้สั่งให้อ่านเรคอร์ดแรกเป็นอันดับแรก โปรแกรมอาจอ่านตรงกลางแฟ้มข้อมูลก่อน การอัพเดตข้อมูลในจานแม่เหล็กอาจเกิดขึ้นไปบางส่วนแล้วจึงจะเกิดสถานะที่บอกว่า *"ข้อมูลทั้งหมดได้ขึ้นมาอยู่ในดิสค์แคชเมมโมรี"* ปัญหาแบบนี้ต้องแก้ด้วยการมีเครื่องทำไฟต่อเนื่อง (Uninterrupted Power Supply) แต่มันก็แก้ได้ไม่หมด เพราะระบบควบคุมที่เรามีกันอยู่นั้นมันยังไม่ฉลาดพอที่จะบอกได้ว่าควรจะคายข้อมูลจากดิสค์แคชเมมโมรีลงในจานแม่เหล็กหรือไม่เมื่อเราสั่งให้บูตเครื่องกันใหม่ บางยี่ห้อก็ทำ บางยี่ห้อก็ไม่ทำ และนี่คือสาเหตุที่ผมบอกว่าเทคนิคที่ผมใช้อยู่นี้มันอาจไม่ทำงานในบางเครื่อง



พูดมาตั้งนาน ผมเกือบลืมชี้ให้เห็นปัญหาขั้นพื้นฐานอีกอย่างหนึ่ง คือ การจัดเก็บข้อมูลเอาไว้ในจานแม่เหล็กนั้นเราต้องมีทะเบียนชื่อ (Disk Directory) เจ้าทะเบียนชื่ออันนี้อยู่ภายใต้การควบคุมของระบบควบคุม มันจะเลือกใช้เนื้อที่จำนวนหนึ่งในแผ่นจานแม่เหล็ก โดยทั่ว ๆ ไปแล้วจะอยู่ที่เซกเตอร์ข้างในสุด ในทะเบียนชื่อนี้มันจะเก็บชื่อของแฟ้มข้อมูล ตำแหน่งที่อยู่ของข้อมูล ลักษณะของแฟ้มข้อมูล ความยาวของเรคอร์ด ตำแหน่งสุดท้ายของแฟ้มข้อมูล (End of File Pointer) และคุณสมบัติอื่น ๆ เท่าที่จำเป็นเอาไว้ ในกรณีที่มีการอัพเดตแบบต้องขยายเนื้อที่เพิ่มเติมไปบางครั้งแล้วนั้น ตำแหน่งที่อยู่ของข้อมูลจริง ๆ อาจกระจายไปอยู่หลายแห่งบนจานแม่เหล็ก กรณีนี้ก็ต้องมีตัวโยงเพื่อใช้ในการอ่านและอัพเดตในภายหลัง เรื่องเหล่านี้ทำโดยโปรแกรมควบคุมระบบ ข้อมูลเหล่านี้จะถูกอ่านแล้วนำไปบันทึกไว้ในเมมโมรี แล้วตัวโปรแกรมควบคุมระบบก็จะคอยแก้ไขปรับปรุงไปตามการทำงานของโปรแกรมของเรา เช่น ถ้าโปรแกรมของเราสั่งบันทึกแบบซีเควนเชียลไปเรื่อย ๆ มันก็จะต้องปรับแก้ตัวชี้ตำแหน่งสุดท้ายของแฟ้มข้อมูลตามไปด้วย และถ้าต้องขยายเนื้อที่ไปกินส่วนอื่นของแผ่นจานแม่เหล็ก มันก็ต้องจัดสร้างตัวเชื่อมโยงขึ้นมา...ฯลฯ ทั้งหมดนี้บางยี่ห้อจะทำในเมมโมรีเท่านั้น มันจะรอจนถึงตอนที่โปรแกรมวิ่งไปถึงคำสั่งให้จบโปรแกรมมันจึงจะบันทึกข้อมูลต่าง ๆ กลับลงไปที่ทะเบียนชื่อแฟ้มข้อมูล แต่บางยี่ห้อก็ทำครึ่ง ๆ กลาง ๆ คือ เมื่อมีการขยายเนื้อที่แต่ละครั้งก็จะปรับแก้ข้อมูลในจานแม่เหล็กด้วย แต่อย่างไรก็ตาม ถ้าโปรแกรมหยุดไปก่อนถึงจุดจบที่แท้จริง ข้อมูลที่เกี่ยวกับตัวแฟ้มข้อมูลเหล่านี้ก็อาจหายไปด้วย ผลที่ตามมาก็คือ ข้อมูลที่เราเติมเข้าไปในการวิ่งงานครั้งนั้นบางอย่างก็มีปรากฏ และบางอย่างก็จะไม่สามารถเรียกกลับออกมาได้ พวกแรกนั้นเป็นพวกที่เราอัพเดตเรคอร์ดในเนื้อที่เก่าที่มีอยู่แล้ว ส่วนพวกหลังนั้นเป็นพวกที่เพิ่มเติมเข้าไปต่อท้ายจากของเดิม พวกที่เราเพิ่มเติมต่อท้ายไปเหล่านี้จริง ๆ แล้วยังมีอยู่ในเนื้อจานแม่เหล็ก (ถ้าเราไม่ได้วิ่งโปรแกรมอื่นเลย) แต่เนื่องจากตัวชี้

ตำแหน่งที่อยู่ของมันยังไม่มีปรากฏอยู่ในทะเบียนชื่อแฟ้มข้อมูล การวิ่งโปรแกรมครั้งใหม่จึงไม่สามารถจะมองเห็นหรือเข้าถึงได้ เนื้อที่ของจานแม่เหล็กที่เก็บเรคอร์ดที่ต่อเติมไปนี้จะอยู่ในสถานะ *"ว่างเปล่า"* หรือ *"พร้อมที่จะให้รองรับข้อมูลใหม่"* (Available) ทั้ง ๆ ที่จริง ๆ แล้วมีข้อมูลบันทึกอยู่

แล้วจะแก้กันอย่างไร

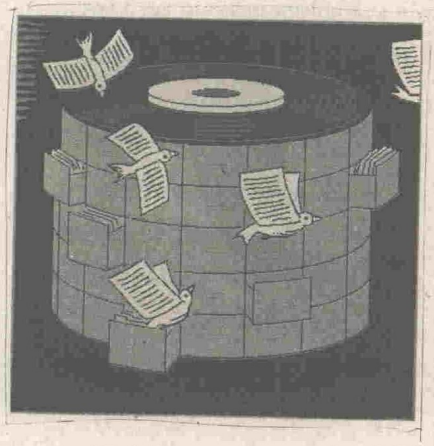
ก็ต้องอ่านกันต่อไปซิครับ แต่ก่อนจะพูดถึงทางออก (Solution) ผมอยากจะชี้เสียนิดหนึ่งว่า พวกเราทุกคนต่างก็ไม่ได้อยากให้ไฟฟ้าดับ หรือทำให้โปรแกรมหยุดไปโดยกะทันหัน แต่มันมีสาเหตุอยู่เหมือนกัน เรื่องของไฟฟ้าดับนั้นเกิดไม่บ่อยนัก คือปีละ 2-3 ครั้ง ซึ่งเราบังคับไม่ได้ ส่วนเรื่องของการหยุดโปรแกรมโดยกะทันหันนั้นเราน่าจะควบคุมได้ แต่ในทางปฏิบัติแล้วเราก็พบว่ามีเหตุสุดวิสัยอยู่เหมือนกัน เช่น พนักงานที่ทำงานค้างไว้แล้วไปทำธุระโดยไม่ระมัดระวัง เมื่อมีเพื่อนพนักงานอีกคนหนึ่งต้องการใช้เครื่องก็อาจเล่นวิธีง่าย ๆ คือบูตเครื่องใหม่ หรือไม่ก็ปิดและเปิดสวิตช์ไฟเข้าเครื่อง หรือแม้แต่เจ้าตัวที่ปล่อยงานค้างไว้นั้นก็อาจสร้างปัญหา คือวิ่งกลับมาแล้วปิดสวิตช์ไฟไปเลย ทั้งนี้เพราะจะรีบกลับบ้าน เป็นต้น ตัวเราเองผู้เขียนโปรแกรมก็ทำอย่างนั้นอยู่บ่อย ๆ ระบบงานที่ดีจึงต้องออกแบบแฟ้มข้อมูลของเราให้ทนต่อสภาวะแบบนั้น นอกจากนี้แล้วเราก็ยังมีกรณีที่โปรแกรมติดลูปหรือผู้ใช้โปรแกรมหาทางออกไม่ถูก เราก็ต้องใช้วิธีบูตใหม่ ผมพบกับโปรแกรมสำเร็จรูปหลายชุดแล้วที่ทดลองวิ่งดูแล้วหาทางออก (Exit to end the program) ไม่เจอ เขาต้องกดแป้นพิเศษพิศดารมาก เราขี้เกียจค้นดูในคู่มือก็เลยต้องจบโปรแกรมโดยกะทันหัน คือถอดปลั๊กไฟฟ้าเสียเลย โปรแกรมที่เราสร้างขึ้นมาก็อาจไปสร้างปัญหาให้กับผู้ใช้ในแบบเดียวกันนี้ได้ แต่ถ้าระบบแฟ้มข้อมูลของเราใช้การได้ดี ผู้ใช้ก็จะไม่ด่ามากนัก

หลักการใหญ่

จริง ๆ แล้วปัญหาที่ต้องระวังมีอยู่ 2 อย่าง อย่างแรกคือการที่เรคอร์ดใหม่ ๆ ที่เพิ่มเติมเข้าไปในแฟ้มจะหายเพราะข้อมูลในทะเบียนชื่อ (Directory) ยังไม่ได้อัพเดต ส่วนเรื่องที่ 2 คือการที่ข้อมูลของเราเองเกิดการไม่สอดคล้องหรือยันกัน ผมคิดว่าเรื่องเหล่านี้แก้ไม่ได้ด้วยการบังคับให้เครื่องทำการบันทึกลงแผ่นจานแม่เหล็กจริง ๆ ทุกครั้งที่สั่ง ดังนั้นคำสั่ง FLUSH ในภาษาปาสคาลนั้นผมไม่แนะนำให้ใช้ มันไม่ได้ช่วยอะไรเรามากนัก ข้อแนะนำของผมมีดังนี้

(ก) ใช้แฟ้มข้อมูลแบบ Random และเป็น I/O เท่านั้น

(ข) สร้างแฟ้มโดยเตรียมเรคอร์ดเอาไว้ให้เพียงพอต่อการใช้งานในระยะยาว ๆ และเตรียมโปรแกรมไว้สำหรับการจัดขยายแฟ้มโดยเฉพาะ



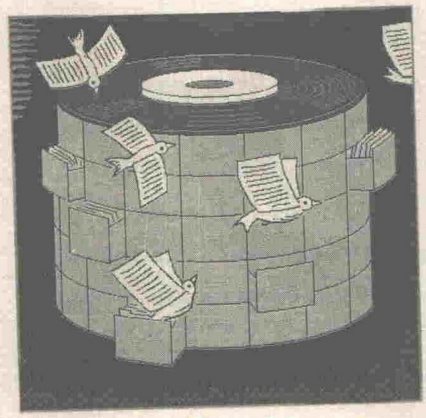
(ค) ในกรณีที่มีเรคอร์ดหลายชนิด เราต้องอัพเดตตัวชี้บอก Next Available Record หลังสุด

(ง) พยายามคิดอะไร ๆ ให้ง่ายเอาไว้

ผมจะขออธิบายความหมายของหลักการต่าง ๆ ข้างบนนี้และชี้ถึงประโยชน์ของมันไปทีละอย่าง จะขอเริ่มจากการใช้แฟ้มข้อมูลแบบแรนด้อม (Random) และเป็นชนิดมีทั้งให้อ่านและบันทึกได้ (I/O File) ทุกครั้งเสียก่อนแฟ้มข้อมูลในคอมพิวเตอร์ขนาดใหญ่นั้นเขามักจะแบ่งออกเป็นแบบซีเควนเชียล (Sequential), อินเดกซ์ซีเควนเชียล (Indexed Sequential) และแรนด้อม (Random) แบบแรกนั้นอ่านเรียงลำดับไป บันทึกก็เรียงลำดับไป การเปิดแฟ้มแบบอินพุต (Input) จะบังคับว่าต้องมีแฟ้มชื่อนั้นอยู่แล้ว ส่วนการเปิดแฟ้มแบบเอาต์พุต (Output) จะไม่บังคับว่าต้องมีแฟ้มชื่อนั้นแล้วหรือยัง แต่ผลของการเปิดแฟ้มแบบเอาต์พุตคือเครื่องจะหาเนื้อที่ในจานแม่เหล็กให้ใหม่ และจะบันทึกข้อมูลหรือเรคอร์ดของเราลงในเนื้อที่ใหม่ดังกล่าว เมื่อไรเราจบโปรแกรมตามปกติแล้ว โปรแกรมควบคุมระบบก็จะขึ้นทะเบียนชื่อแฟ้มข้อมูลอันนี้ให้เรา ในกรณีที่มีแฟ้มชื่อเดียวกันอยู่ในจานแม่เหล็กอยู่แล้ว มันก็จะยกเลิกข้อมูลในแฟ้มเก่าและลบชื่อเก่าทิ้งไป โดยสรุปแล้วการใช้แฟ้มข้อมูลแบบซีเควนเชียลนี้ทำให้เกิดการย้ายที่อยู่ของแฟ้มเมื่อเปิดแบบเอาต์พุต (แต่ไม่มีการย้ายที่อยู่เมื่อเปิดใช้แบบอินพุต) นอกจากนี้แล้ว การเปิดแฟ้มแบบเอาต์พุตนั้นจะทำให้ต้องใช้เนื้อที่เป็นประมาณ 2 เท่าของขนาดแฟ้ม ทั้งนี้เพราะของเก่าก็ยังอยู่บนจานในขณะที่กำลังสร้างของใหม่

การใช้แฟ้มแบบซีเควนเชียลจึงเปลืองที่ และมีความเสี่ยงที่จะเกิดปัญหาในระหว่างการทำงาน เช่น ถ้าจานแม่เหล็กเกิดเต็มขึ้นมาในระหว่างทำงานก็จะเกิดการจบโปรแกรมโดยกะทันหัน ข้อมูลใหม่ทั้งหมดจะหายไป หรือถ้าใครเดินมาเตะปลั๊กไฟฟ้าก็จะทำให้ข้อมูลใหม่หายไปเช่นกัน การใช้แฟ้มข้อมูลแบบซีเควนเชียลนี้ดูจะง่าย แต่มันจะทำงานช้า และมีความเสี่ยงสูง ผมจึงไม่แนะนำให้ใช้

การจัดแฟ้มแบบอินเดกซ์ซีเควนเชียลนั้นผมหมายถึงการใช้ยูทิลิตี้ที่มากับซิสเต็มส์ซอฟต์แวร์กันเลย หรือคือซิสเต็มส์ซอฟต์แวร์จะจัดการสร้างตัวดัชนี (Index) ให้เราโดยอัตโนมัติและจะขยายเนื้อที่ให้เราโดยอัตโนมัติ เรื่องนี้มองดูก็ง่ายดีและเนื้อที่ในจานแม่เหล็กก็ประหยัดดี คือมีก็อบปี้เดียวอยู่ในจานแม่เหล็ก เราสามารถเปิดแฟ้มใช้ได้ทั้งแบบเป็นอินพุตหรือเอาต์พุตอย่างเดียว หรือจะทำทั้งอินพุตและเอาต์พุตก็ได้ แต่เรคอร์ดที่เพิ่มเติมเข้ามาจะอยู่กระจัดกระจายออกไปในแผ่นจานแม่เหล็ก ตัวยูทิลิตี้จะเรียกหาเนื้อที่จานแม่เหล็กเมื่อต้องเก็บเรคอร์ดเพิ่มขึ้น มันจึงมีโอกาสที่จะติดปัญหาเมื่อจานแม่เหล็กไม่มีที่ว่างอีกแล้ว นอกจากนี้แล้วเมื่อไฟฟ้าเกิดขาดลงระหว่างการทำงาน ข้อมูลที่เป็นเรคอร์ดใหม่เหล่านี้จำนวนหนึ่งจะหายไป แต่ตัวดัชนีอาจมีแล้วก็ได้ แฟ้มแบบนี้จึงยิ่งไม่เหมาะ เพราะเราไปแก้ไขตัวดัชนีของเขาไม่ได้ ผู้ใช้เครื่องเมนเฟรมต้องแก้ปัญหาด้วยการบังคับให้มีระบบไฟฟ้าชนิดต่อเนื่อง (UPS) เทคนิคนี้ไม่ค่อยจะเหมาะกับเครื่องระดับไมโครคอมพิวเตอร์ เราจึงไม่ค่อยได้เห็นซิสเต็มส์ซอฟต์แวร์ที่ใช้แบบอินเดกซ์ซีเควนเชียลกัน และถึงแม้จะซื้อหามาได้ผมก็ไม่



แนะนำให้ใช้ เพราะมันมีโอกาสตายได้เมื่อถึงตอนนั้นท่านจะโทรไปขอให้เขามาแก้ไขปัญหาให้ก็คงจะยาก

สรุปแล้วก็เหลือเพียงการจัดแฟ้มแบบแรนด้อม แฟ้มแบบนี้ผมหมายถึงแฟ้มที่เวลาสั่งให้อ่านหรือบันทึกก็จะต้องบอกหมายเลขตำแหน่งที่ของเรคอร์ด (Relative Key) ในภาษาบางภาษา เช่น ปาสคาลนั้นต้องใช้วิธีสั่ง SEEK เสียก่อนแล้วจึงจะสั่ง READ หรือ WRITE คำสั่ง SEEK จะต้องบอกเลขลำดับที่ของเรคอร์ด เช่น

SEEK (FILEA, 15);

ซึ่งหมายถึงว่า ให้วิ่งไปที่เรคอร์ดที่ 15 ของ FILEA

แฟ้มแบบแรนด้อมนี้มีข้อดีอยู่ที่ว่าไม่ว่าเราจะเปิดแฟ้มเป็นแบบอินพุตอย่างเดียวหรือเอาต์พุตอย่างเดียว หรือเป็นทั้งอินพุตและเอาต์พุต มันก็จะใช้แฟ้มที่มีอยู่แล้วอันนั้น และมันเป็นระบบที่ค่อนข้างตรงไปตรงมา คืออยากได้เรคอร์ดที่เท่าใดก็ต้องบอกตำแหน่ง (ลำดับที่) ของเรคอร์ดนั้น แล้วจึงจะสั่งให้อ่านหรือบันทึกลงไป ปัญหาที่ต้องระวังก็เหลือแต่เรื่องการขยายของเนื้อที่ที่จะใช้ ซึ่งเราจะแก้ได้ด้วยหลักการข้อ (ข)

ครับ หลักการข้อ (ข) นั้นผมหมายถึงว่า เราจะต้องสร้างแฟ้มข้อมูลเอาไว้ครั้งแรกโดยเคลียร์พื้นที่เอาไว้จำนวนหนึ่ง โดยให้มีจำนวนเรคอร์ดไว้มากพอสำหรับใช้เป็นระยะยาว คำว่า *"เคลียร์พื้นที่"* นี้ผมหมายความว่า ให้เปิดแฟ้มเป็นเอาต์พุต แล้วเอาเรคอร์ดที่ว่างเปล่าบันทึกลงไปไว้เป็นจำนวนที่ต้องการ แล้วสั่งปิดแฟ้มข้อมูลนั้น เรื่องนี้ควรทำเป็นโปรแกรมเล็ก ๆ เพื่อกิจกรรมนี้โดยเฉพาะ เราวิ่งโปรแกรมนี้ครั้งเดียว หลังจากนั้นก็ใช้แฟ้มนี้ไปเรื่อย ๆ จนกว่าจะเต็ม และเพื่อให้เรารู้ว่าเราได้ใช้แฟ้มข้อมูลนี้ไปกี่เรคอร์ดแล้ว โปรแกรมนี้ก็จะต้องบันทึกตัวเลขอันนี้เอาไว้ที่ไหนสักแห่งหนึ่ง ที่ง่ายที่สุดก็คือเก็บที่เรคอร์ดแรก เรคอร์ดนี้จะไม่ใช้เก็บข้อมูล ในภาษาปาสคาลนั้นจะมีความประจวบเหมาะอยู่อย่างหนึ่ง คือเรคอร์ดแรกนั้นเขาเรียกว่าเรคอร์ดที่ 0 (เลขศูนย์) เรคอร์ดที่ 2 กลับเรียกว่า เรคอร์ดที่ 1... ฯลฯ ดังนั้นเรคอร์ดที่จะเป็นตัวข้อมูลจะตรงกับเรคอร์ดที่ 1 เป็นต้นไป

ในการเคลียร์พื้นที่ดังกล่าวมาข้างต้นนี้ เราควรทำในตอนที่จานแม่เหล็กของเราจัดเก็บแฟ้มข้อมูลอื่นไว้อย่างอัดแน่น คือมีที่ว่างอยู่ตอนท้ายเป็นชิ้นใหญ่เพียงชิ้นเดียว เรื่องนี้เราทำได้โดยก็อบปี้แฟ้มข้อมูลทั้งหมดออกไปไว้ที่อื่น ทำการฟอร์แมตจานแม่เหล็กเสียใหม่ แล้วก็อบปี้แฟ้มต่าง ๆ กลับลงไป ด้วยวิธีนี้แฟ้มข้อมูลเหล่านี้จะอยู่เรียงชิดติดกันเป็นพืดแล้วมีที่ว่างเหลืออยู่ตอนท้ายเป็นผืนเดียว เมื่อเราวิ่งโปรแกรมเพื่อสร้างและเคลียร์พื้นที่ให้กับแฟ้มข้อมูลใหม่ เรคอร์ดทั้งหมดจะอยู่ชิดและต่อเนื่องกันไป แล้วแฟ้มข้อมูลอันนี้ก็จะอยู่ในตำแหน่งดังกล่าวตลอดไป และอยู่ไป

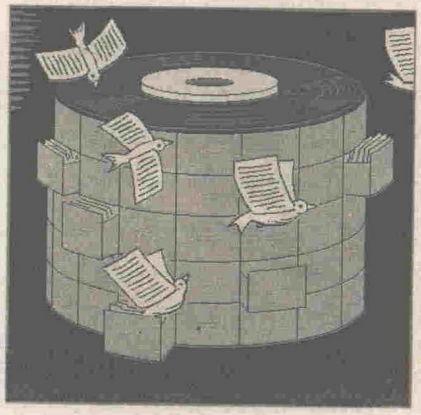
จนกว่าจะมีการเลิกใช้หรือขยายเนื้อที่กันใหม่

นอกจากการเคลียร์พื้นที่ และการเก็บตัวเลขบอกจำนวนเรคอร์ดที่ใช้ไปแล้ว *(ในทางปฏิบัติจริง ๆ นั้นเราจะเก็บตัวเลขที่จะชี้ไปยังเรคอร์ดที่จะใช้งานถัดไป ซึ่งเรียกว่า Next Available Record Number ค่านี้จะมากกว่า "จำนวนเรคอร์ดที่ใช้ไปแล้ว" อยู่ 1)* เราควรที่จะให้มีฟิลด์หนึ่งในทุกเรคอร์ดเพื่อบอกสถานะของเรคอร์ด สถานะของเรคอร์ดจะมีได้ 3 สถานะคือ ยังว่างเปล่า ใช้อยู่ และเลิกใช้แล้ว เราอาจใช้รหัสว่า

0 = ว่างเปล่า = Available
1 = ใช้อยู่ = Active
2 = เลิกใช้แล้ว = Deleted

ฟิลด์นี้อาจให้อยู่หน้าสุดของแต่ละเรคอร์ด และใช้เพียง 1 ไบต์เท่านั้น รายละเอียดของโปรแกรมที่จะใช้สร้างและเคลียร์พื้นที่ให้แก่เรานี้จะอยู่ในหัวข้อถัดไป สิ่งที่ผมอยากจะชี้ให้เห็นในตอนนี้ก็คือ *จริง ๆ แล้วการวิ่งโปรแกรมเพื่อสร้างเรคอร์ดที่มีสถานะเป็น "0 = วิ่งเปล่า = Available" เอาไว้จำนวนหนึ่งนี้ ในสายตาของตัวโปรแกรมควบคุมระบบ มันหมายถึงการใช้เนื้อที่ในจานแม่เหล็กไปแล้ว* หรือคือถ้าเราเคลียร์เอาไว้ 3000 เรคอร์ด เจ้าตัวควบคุมระบบก็จะบันทึกเอาไว้ว่าตัวชี้ไปยังท้ายของแฟ้ม (End of File Pointer) ได้ชี้ไปที่เลข 3000 ไปแล้ว เรคอร์ดเหล่านี้จะมีอยู่จริง ๆ และจะไม่มีวันหายไปไหนอีกแล้ว ถ้าโปรแกรมอัพเดตของเราจะไปตายกลางทาง หรือเมื่อเกิดไฟฟ้าดับขึ้นมาในภายหลัง เจ้าตัวควบคุมระบบก็ยังยืนยันว่าเรามีเรคอร์ดอยู่ 3000 เรคอร์ด เรคอร์ดเหล่านี้จึงไม่มีวันจะสูญหายไปได้ (เชื่อได้ 99.99... เปอร์เซ็นต์) นี่คือจุดมุ่งหมายที่เป็นที่มาของบทความนี้

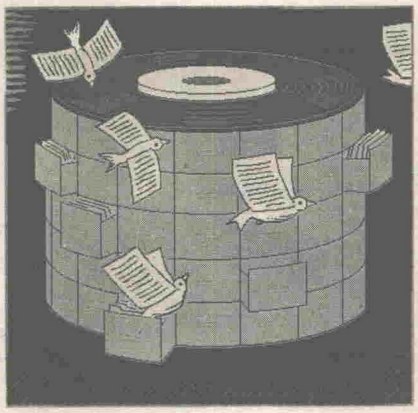
ข้างต้นนั้นเป็นหลักการที่จะทำให้เรคอร์ดไม่หายไป แต่มันยังไม่ได้การันตีว่าข้อมูลที่อยู่ภายในแฟ้มเหล่านี้จะยันกันได้ตลอด เรื่องนี้ต้องอาศัยหลักการข้อ (ค) แต่เราการันตีได้ไม่ถึง 100% หลักการข้อ (ค) บอกว่าให้อัพเดตเรคอร์ดประเภทตัวชี้บอก Next Available Record หลังสุด เรื่องนี้จริง ๆ แล้วเรามีเรคอร์ดอยู่ 3 ชนิดด้วยกัน เรคอร์ดประเภทแรกคือเรคอร์ดที่เก็บข้อมูล ซึ่งอาจเรียกว่า Master File Record เรคอร์ดประเภทที่ 2 คือเรคอร์ดที่เก็บดัชนี ซึ่งเป็นตัวชี้บอกตำแหน่งที่อยู่ของเรคอร์ดประเภทแรก เรคอร์ดเก็บดัชนีนี้อาจมีหลายระดับ คือแยกเป็นการชี้กลุ่มเสียก่อน แล้วในกลุ่มจึงจะชี้ไปยังเรคอร์ดประเภทแรก เรคอร์ดที่เก็บดัชนีนี้จะออกแบบให้เป็นส่วนหนึ่งของแฟ้มข้อมูลหลัก หรือจะแยกไปเก็บเป็นแฟ้มต่างหากก็ได้ แต่แนะนำว่าให้จัดไว้โดยมีจำนวนที่แน่นอน เรคอร์ดประเภทสุดท้ายคือ เรคอร์ดที่เก็บตัวเลขบอกว่าเราใช้เนื้อที่แฟ้มไปแล้วเท่าใด ตัวเลขนี้มี 2 ตัวคือ Next Available Record และ Last Master Record การบันทึกเรคอร์ดต่าง ๆ ทั้ง 3 ชนิดนี้ให้ทำเรียงลำดับจาก 1 มา 3



ด้วยวิธีดังกล่าวมาข้างต้นนี้เราจะเห็นว่า ถ้าไฟฟ้าดับหรือโปรแกรมหยุดก่อนที่จะมีการอัพเดตค่าในเรคอร์ดประเภทที่ 3 ก็จะทำให้ระบบของเรายังไม่ยอมรับเรคอร์ดใหม่ที่เพิ่มขึ้น แต่ระบบจะยังทำงานได้เมื่อวิ่งโปรแกรมกันใหม่ มีสิ่งที่ต้องระวังอยู่อย่างเดียวคือในกรณีที่มีการอัพเดตแบบเติมเรคอร์ดใหม่เข้าไป และไฟฟ้าไปดับในจังหวะที่กำลังอัพเดตเรคอร์ดประเภทดัชนีพอดี ในกรณีนี้จะมีจำนวนดัชนีมากกว่าจำนวนเรคอร์ดอยู่ 1 ตัว โอกาสที่จะเกิดกรณีนี้มีไม่มาก เพราะการอัพเดตดัชนีกับการอัพเดตตัวชี้ไปยัง Next Available Record จะอยู่กระชั้นชิดกันมาก แต่ก็มีเกิดขึ้นได้ เรื่องนี้ผมแนะนำว่าให้ทำโปรแกรมไว้อีก 1 โปรแกรม ทั้งนี้เพื่อใช้เช็คยันจำนวนดัชนีกับจำนวนเรคอร์ดในแฟ้มข้อมูลหลัก และถ้าเห็นว่าผิดปกติก็จะทำการสร้างแฟ้มดัชนีขึ้นใหม่

ขอผ่านมาถึงหลักการข้อสุดท้าย หลักการข้อ (ง) นี้เปิดกว้างมาก คือกล่าวว่า *"ให้พยายามคิดอะไร ๆ ง่าย ๆ"* ความหมายของผมก็คือ อย่าไปเตรียมการเพื่อแก้ปัญหาเอาไว้ล่วงหน้า เช่น อย่าพยายามเขียนเอาไว้ในโปรแกรมอัพเดตว่าจะทำอย่างไรเมื่อแฟ้มข้อมูลเต็ม หรือเมื่อเรคอร์ดที่เก็บดัชนีบางกลุ่มมันเต็ม สิ่งเหล่านี้คิดไปเท่าไรก็ไม่มีวันจบสิ้น วิธีที่ดีที่สุดคือให้ออกคำเตือนว่า "File is full" หรือ "Index record is full" แล้วยกเลิกการอัพเดตอันนั้นไปเลย ในกรณีเช่นนี้แฟ้มข้อมูลยังถูกต้องดีอยู่ แล้วเราต้องมีโปรแกรมเล็ก ๆ อีก 1-2 โปรแกรม เพื่อใช้สร้างหรือขยายขนาดของแฟ้ม การแยกมาทำด้วยโปรแกรมย่อยเล็ก ๆ อย่างนี้จะลดโอกาสผิดพลาดหรือบกพร่อง

และมีโอกาสทดสอบดูความเรียบร้อยเสียก่อนที่โปรแกรมทั้งระบบอาจต้องนำมาแก้ไขและคอมพายล์ใหม่ แต่ถ้าเรายึดหลักการ *"ทำให้ง่าย ๆ"* ไว้ได้จริง ๆ โปรแกรมเหล่านี้ก็จะมีจุดที่ต้องแก้ไขเพียงนิดเดียว เช่น จำนวนดัชนีต่อ 1 เรคอร์ด เป็นต้น



ตัวอย่างโปรแกรม

ผมจะขอยกตัวอย่างเป็นภาษาปาสคาล และจะเขียนให้อยู่ในลักษณะที่ใช้งานได้จริง ๆ ทั้งนี้เพื่อย้ำว่า โปรแกรมเหล่านี้จะสั้นและเข้าใจง่าย และโอกาสพลาดมีน้อย ในตัวอย่างนี้ผมจะสมมติให้เป็นเรื่องทะเบียนรถยนต์ของธนาคารขนาดกลางแห่งหนึ่ง ธนาคารนี้มีรถยนต์อยู่ประมาณ 300 คัน อัตราการเพิ่มขึ้นปีละไม่ถึง 10% แต่เราจะเคลียร์พื้นที่เอาไว้ 900 เรคอร์ด ซึ่งจะเกินพอสำหรับ 5 ปีข้างหน้า แฟ้มข้อมูลอันนี้จะเก็บอะไรบ้างนั้นคงจะไม่สำคัญ ผมจะเตรียมที่ไว้ให้ 45 ตัวอักษร แต่สิ่งที่ขาดไม่ได้ก็คือ *"หมายเลขทะเบียนรถ"* ซึ่งจะใช้เป็นคีย์ (Key) หรือเลขประจำตัว (ID) เพื่อการเรียกหาข้อมูล และเพื่อให้สามารถเรียกดูข้อมูลได้อย่างรวดเร็ว ผมจะจัดระบบให้มีดัชนีเอาไว้ด้วย ดัชนีชุดนี้ไม่จำเป็นต้องจัดเรียง ในกรณีที่ต้องการทำรายงานที่จัดเรียงตามหมายเลขทะเบียนรถ เราจะอ่านเอาดัชนีทั้งหมดขึ้นมาใส่ในเมมโมรี แล้วจัดเรียงด้วยรูทีนภายในโปรแกรม ซึ่งจะใช้เวลาไม่ถึง 1 นาที แล้วออกรายงานได้เลย แต่ดัชนีชุดนี้จะต้องทำให้เราเข้าถึงข้อมูลโดยบอกหมายเลขทะเบียนได้โดยง่าย วิธีการที่ง่ายที่สุดก็คือ ให้เอาตัวอักษรและตัวเลขที่ปรากฏอยู่ในหมายเลขทะเบียนไปแปลเป็นค่าของรหัสแอสกี (ASCII Code) ทีละตัว แล้วนำมาบวกกัน ซึ่งจะให้ผลเป็นตัวเลขจำนวนเต็ม (Integer) ตัวหนึ่ง เมื่อได้แล้วก็ตัดเอาแค่ 2 หลักท้ายแล้วเอาเลข 1 บวกเข้าไป ผลที่ได้ก็จะมีค่าตั้งแต่ 1 ถึง 100 แล้วใช้เลขดังกล่าวเป็นตัวบอกกลุ่ม หรือคือเป็นตัวชี้ไปยังเรคอร์ดที่เก็บดัชนี ด้วยวิธีนี้เราคาดว่าดัชนีจะกระจายไปอยู่ในเรคอร์ดต่าง ๆ 100 เรคอร์ดอย่างใกล้เคียงกัน

จากการกำหนดวิธีการจัดดัชนีดังกล่าวมาข้างต้นนี้ เราจะมีเรคอร์ดดัชนีอยู่ 100 เรคอร์ด แต่ละเรคอร์ดจะมีดัชนีอยู่ประมาณ 900/100 = 9 ดัชนี แต่เพื่อไม่ให้เกิดการเต็มไปก่อนกำหนด เราควรเตรียมไว้สัก 2 เท่า คือให้มีเรคอร์ดละ 20 ดัชนี แต่ละดัชนีจะประกอบไปด้วยหมายเลขทะเบียนรถซึ่งยาว 8 ตัวอักษร กับตัวเลขบอกตำแหน่งที่อยู่ของเรคอร์ดที่เก็บข้อมูลซึ่งยาว 3 หลัก ดังนั้นเรคอร์ดของแฟ้มดัชนีจะยาวเรคอร์ดละ 20×13 ไบต์ เราจะเก็บดัชนีไว้ในรูปของ "String" ซึ่งจะทำให้สามารถเรียกออกมาดูได้ด้วยคำสั่ง TYPE ของดอส

จากข้อกำหนดข้างต้น ผมก็สามารถเขียนโปรแกรมเพื่อสร้างแฟ้มข้อมูลและเคลียร์พื้นที่ไว้ได้ โปรแกรม FT1S01.PAS ในรูปที่ 1 คือตัวอย่างดังกล่าวนี้ โปรแกรมนี้ใช้เวลาเขียน 1 ชั่วโมง และวิ่ง 1 นาทีก็จบ

หลังจากมีโปรแกรมสร้างแฟ้มข้อมูลและแฟ้มดัชนีแล้ว เราก็ต้องทำโปรแกรมเพื่อการอัพเดตข้อมูล การอัพเดตจะมี 3 ลักษณะคือ

(ก) เพิ่มเรคอร์ดใหม่ (Add New Record)

(ข) แก้ไขข้อมูลในเรคอร์ดเก่า (Change Data)

(ค) ยกเลิกเรคอร์ดใด ๆ (Delete Record)

โปรแกรมนี้คือโปรแกรมที่จะปฏิบัติตามหลักการข้อ (ง) ในหัวข้อที่ผ่านมา คือจะทำอะไรก็ให้คิดง่าย ๆ เช่น การเพิ่มเรคอร์ดใหม่นั้นจะเช็คว่าหมายเลขทะเบียนรถใหม่ซ้ำกับของเก่าหรือไม่ ถ้าไม่ซ้ำก็จะมารับข้อมูลเข้าไป และพยายามจะเติมเรคอร์ดใหม่ลง ณ จุดเรคอร์ดที่ว่างเปล่าถัดไป (Next Available Record) หากเติมไม่ได้เพราะไฟล์เต็มหรือเรคอร์ดที่จะเก็บดัชนีเต็มแล้ว เราก็จะบอกสาเหตุแล้วจบโปรแกรมไปเลย ถ้าอัพเดตได้ก็จะขึ้นข้อความบอกให้รู้ว่าเสร็จเรียบร้อย (Complete) ในการหาจุดที่จะเติมดัชนีลงไปนั้นก็จะไล่ดูทีละช่อง จนกว่าจะเจอช่องที่ตัวชี้ (Pointer) มีค่าเป็น "000" เราจะไม่มีการสร้างเรคอร์ดดัชนีใหม่ขึ้นมา เพราะการทำเป็น Link-list นั้นยุ่งยาก สำหรับการยกเลิกเรคอร์ดนั้นก็เช่นกัน เราจะค้นหารถหมายเลขทะเบียนดังกล่าวแล้วนำข้อมูลขึ้นมาแสดงบนจอภาพ แล้วถามว่าใช่หรือไม่ ถ้าตอบว่าใช่ (Y) ก็จะจัดการเปลี่ยนรหัสสถานะของเรคอร์ดเป็นเลข "2" และเปลี่ยนตัวดัชนีให้กลับว่างเปล่า ทั้งนี้เพื่อให้สามารถใช้เนื้อที่ตรงนี้เก็บดัชนี

รูปที่ 1 โปรแกรม FT1S01.PAS เพื่อสร้างไฟล์

```
PROGRAM FT1S01;
LABEL CLEARINDEXFILE,CLEARMASTERFILE,EOJ;
CONST MAXINDX=20;MAXINDXRECORD=100;MAXMASTER=900;
TYPE  STR3=STRING[3];
      STR8=STRING[8];
      STR45=STRING[45];
TYPE  INDEXRECORD=RECORD
      KEY:ARRAY[1..MAXINDX] OF STR8;
      POINTER:ARRAY[1..MAXINDX] OF STR3;
      END;
TYPE  MASTERRECORD=RECORD
      STATUS:CHAR;
      ID:STR8;
      DATA:STR45;
      END;
VAR INDEXFILE:FILE OF INDEXRECORD;
VAR INDEXBUFF:INDEXRECORD;
VAR MASTERFILE:FILE OF MASTERRECORD;
VAR MASTERBUFF:MASTERRECORD;
VAR RECNO,K,INDXRECNO:INTEGER;
VAR CH:CHAR;

BEGIN
  CLRSCR;
  GOTOXY(1,1);WRITE('PROGRAM FT1S01 IS RUNNING');
  GOTOXY(1,2);WRITE(' OLD MASTER FILE WILL BE WIPPED OUT.');
  GOTOXY(1,3);WRITE(' ARE YOU SURE? PLEASE CONFIRM (Y/N) ');
  REPEAT UNTIL KEYPRESSED;
  READ(KBD,CH);
  IF CH<>'Y' THEN GOTO EOJ;

CLEARINDEXFILE:
  GOTOXY(1,5);WRITE(' CLEARING THE INDEX-FILE ');
  ASSIGN(INDEXFILE,'FT1D01.DAT');
  REWRITE(INDEXFILE);
  FOR K:=1 TO MAXINDX DO
    BEGIN
    INDEXBUFF.KEY[K]:='XXXXXXXX';
    INDEXBUFF.POINTER[K]:='000';
    END;
  FOR INDXRECNO:=0 TO MAXINDXRECORD DO
    BEGIN
    GOTOXY(5,6);WRITE('INDXRECNO=',INDXRECNO);
    SEEK(INDEXFILE,INDXRECNO);
    WRITE(INDEXFILE,INDEXBUFF);
    END;
  CLOSE(INDEXFILE);

CLEARMASTERFILE:
  GOTOXY(1,7),WRITE(' CLEARING THE MASTER-FILE ');
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  REWRITE(MASTERFILE);
  MASTERBUFF.STATUS:='1';
  MASTERBUFF.ID:='00000001';
  MASTERBUFF.DATA:='FIRST MASTER RECORD
  SEEK(MASTERFILE,0);
  WRITE(MASTERFILE,MASTERBUFF);
  MASTERBUFF.STATUS:='0';
  MASTERBUFF.ID:='XXXXXXXX';
  MASTERBUFF.DATA:='




  FOR RECNO:=1 TO MAXMASTER DO
    BEGIN
    GOTOXY(5,8);WRITE('RECNO=',RECNO);
    SEEK(MASTERFILE,RECNO);
    WRITE(MASTERFILE,MASTERBUFF);
    END;
  CLOSE(MASTERFILE);

EOJ:
  GOTOXY(1,10);WRITE('END OF JOB');
  END.
```

ตัวใหม่ได้เลย การบันทึกข้อมูลกลับลงไปในจานแม่เหล็กก็จะเรียงลำดับกัน คือ บันทึกเรคอร์ดที่เก็บข้อมูลก่อน แล้วจึงบันทึกแก้ดัชนี และถ้าจะมีการแก้ไขตัวชี้ไปยัง Next Available Record ก็จะทำทีหลังสุด

โปรแกรม FT1S02.PAS ในรูปที่ 2 คือโปรแกรมเพื่อใช้ในการอัพเดตดังกล่าว ผมได้เติมรูทีนเพื่อการเรียกดูข้อมูลและการสร้างดัชนีให้ใหม่เอาไว้ด้วย โปรแกรมนี้ยาวเพียง 414 บรรทัด ใช้เวลาเขียนและทดสอบ 6 ชั่วโมง หน้าตาของจอภาพอาจยังดูไม่สวยนัก ท่านผู้อ่านต้องเอาไปปรับปรุงเพิ่มเติมเอง เพราะข้อมูลจริง ๆ จะมีกี่ฟิลด์จะขึ้นอยู่กับผู้ใช้งาน

ข้อควรสังเกต

การออกแบบแฟ้มข้อมูลตามที่บรรยายมานี้เราจะเห็นว่าแฟ้มดัชนีนั้นไม่ใช่แฟ้มที่สำคัญ ข้อมูลที่สำคัญจะอยู่ในแฟ้มข้อมูลหลัก ถ้าจำนวนเรคอร์ดในแฟ้มข้อมูลหลักไม่ได้หายไปและเรคอร์ดแรกที่เก็บค่า Next Available Record ยังชี้ได้ถูกต้อง เราก็จะสามารถสร้างดัชนีขึ้นใหม่ได้โดยไม่มีผลเสียหายอะไรเลย แฟ้มดัชนีมีประโยชน์ในการเข้าถึงข้อมูลโดยผ่านทางคีย์ (Key) หรือหมายเลขประจำตัว (ID) เท่านั้น เราจะสร้างขึ้นใหม่เมื่อไหร่ก็ได้ ในโปรแกรมตัวอย่าง (FT1S02.PAS) นี้เราสามารถใช้รูทีน BROWSING เพื่อดูเรคอร์ดต่าง ๆ ในแฟ้มข้อมูลหลักเสียก่อน เรคอร์ดแรก (RECNO=0) จะบอกค่า Next Available Record เรคอร์ดถัด ๆ มาจะเป็นข้อมูลที่เก็บไว้ ถ้าทุกเรคอร์ดอยู่ในสภาพดีเราก็สามารถใช้รูทีน REBUILD INDEX FILE เพื่อสร้างดัชนี

รูปที่ 2 โปรแกรมอัพเดตและเรียกดูข้อมูล FT1S02.PAS

```
OGRAM FT1S02;
BEL MAINLOOP,READKBD,EOJ;
NST MAXINDX=20;MAXINDXRECORD=100;MAXMASTER=900;
PE  STR3=STRING[3];
    STR8=STRING[8];
    STR45=STRING[45];
PE  INDEXRECORD=RECORD
    KEY:ARRAY[1..MAXINDX] OF STR8;
    POINTER:ARRAY[1..MAXINDX] OF STR3;
    END;
PE  MASTERRECORD=RECORD
    STATUS:CHAR;
    ID:STR8;
    DATA:STR45;
    END;
R INDEXFILE:FILE OF INDEXRECORD;
R INDEXBUFF:INDEXRECORD;
R MASTERFILE:FILE OF MASTERRECORD;
R MASTERBUFF,FIRSTMASTERREC:MASTERRECORD;
R RECNO,NEXTAVAILRECORD,INDXRECNO,INDXPOSN:INTEGER;
R LICNO:STR8;
R CARDATA:STR45;
R CH:CHAR;

OCEDURE BEEP;
GIN
SOUND(400):DELAY(100);
OSOUND;DELAY(50);
);

OCEDURE NEWSCREEN;
R K1:INTEGER;
GIN
CLRSCR;
FOR K1:=2 TO 79 DO
  BEGIN
  GOTOXY(K1,1);WRITE('*');
  GOTOXY(K1,25);WRITE('*');
  END;
GOTOXY(35,1);WRITE(' PROGRAM:FT1S02 ');
);

OCEDURE CLEARLINE(N:INTEGER);
GIN
GOTOXY(2,N);
VRITE('                                                    ');
);

OCEDURE CONVERTSTRING(STR:STR3;VAR NUM:INTEGER);
R I,SUM:INTEGER;
R CH:CHAR;
GIN
SUM:=0;
FOR I:=1 TO 3 DO
  BEGIN
  CH:=COPY(STR,I,1);
  SUM:=10*SUM+ORD(CH)-48;
  END;
UM:=SUM;
);

OCEDURE CONVERTNUM(NUM:INTEGER;VAR STR:STR3);
R I,DI:INTEGER;
GIN
TR:='';
FOR I:=1 TO 3 DO
  BEGIN
  DI:=NUM MOD 10;
  STR:=CHR(DI+48)+STR;
  NUM:=NUM DIV 10;
  END;
);

OCEDURE CALCINDXRECNO;
R I,SUM:INTEGER;
R CH:CHAR;
GIN
SUM:=0;
FOR I:=1 TO 8 DO
  BEGIN
  CH:=COPY(LICNO,I,1);
  SUM:=SUM+ORD(CH);
  END;
INDXRECNO:=(SUM MOD MAXINDXRECORD)+1;
);

PROCEDURE GETINDXRECORD;
BEGIN
  SEEK(INDEXFILE,INDXRECNO);
  READ(INDEXFILE,INDEXBUFF);
END;

PROCEDURE SEARCHINDXRECORD;
VAR I:INTEGER;
BEGIN
  INDXPOSN:=0;
  FOR I:=1 TO MAXINDX DO
    IF (LICNO=INDEXBUFF.KEY[I]) THEN INDXPOSN:=I;
END;

PROCEDURE FINDEMPTYINDEX;
VAR I:INTEGER;
BEGIN
  INDXPOSN:=0;
  FOR I:=1 TO MAXINDX DO
    IF (INDEXBUFF.KEY[I]='XXXXXXXX') AND (INDXPOSN=0) THEN
                              INDXPOSN:=I;
END;

PROCEDURE GETMASTERRECORD;
BEGIN
  SEEK(MASTERFILE,RECNO);
  READ(MASTERFILE,MASTERBUFF);
END;

PROCEDURE PUTMASTERRECORD;
BEGIN
  SEEK(MASTERFILE,RECNO);
  WRITE(MASTERFILE,MASTERBUFF);
END;

PROCEDURE PUTINDXRECORD;
BEGIN
  SEEK(INDEXFILE,INDXRECNO);
  WRITE(INDEXFILE,INDEXBUFF);
END;

PROCEDURE GETFIRSTMASTERRECORD;
VAR NXTREC:STR3;
BEGIN
  SEEK(MASTERFILE,0);
  READ(MASTERFILE,FIRSTMASTERREC);
  NXTREC:=COPY(FIRSTMASTERREC.ID,6,3);
  CONVERTSTRING(NXTREC,NEXTAVAILRECORD);
END;

PROCEDURE PUTFIRSTMASTERRECORD;
VAR NXTREC:STR3;
BEGIN
  NEXTAVAILRECORD:=NEXTAVAILRECORD+1;
  CONVERTNUM(NEXTAVAILRECORD,NXTREC);
  FIRSTMASTERREC.ID:='      '+NXTREC;
  SEEK(MASTERFILE,0);
  WRITE(MASTERFILE,FIRSTMASTERREC);
END;

PROCEDURE DISPLAYMASTERRECORD;
BEGIN
  GOTOXY(3,7);WRITE('RECORD-NO = ',RECNO);
  GOTOXY(3,8);WRITE('STATUS    = ',MASTERBUFF.STATUS);
  GOTOXY(3,9);WRITE('LICENCE-NO= ',MASTERBUFF.ID);
  GOTOXY(3,10);WRITE('DATA      = ',MASTERBUFF.DATA);
END;

PROCEDURE WAITTOEXIT;
VAR CH:CHAR;
BEGIN
  CLEARLINE(23);CLEARLINE(24);
  GOTOXY(3,23);WRITE('HIT ANY KEY TO RETURN TO MAIN MENU');
  GOTOXY(3,24);REPEAT UNTIL KEYPRESSED;READ(KBD,CH);
END;

PROCEDURE ACCEPTLICENCENO;
VAR LN:INTEGER;
BEGIN
  GOTOXY(3,23);WRITE('WHAT LICENCE-NUMBER?');
  GOTOXY(3,24);READ(LICNO);
  LN:=LENGTH(LICNO);
  WHILE LN<8 DO
    BEGIN LICNO:=LICNO+' ';LN:=LN+1;END;
END;
```

รูปที่ 2 (ต่อ)

```
PROCEDURE ACCEPTCARDATA;
VAR LN:INTEGER;
BEGIN
  GOTOXY(3,23);WRITE('PLEASE INPUT CAR DATA(45 BYTES)');
  CLEARLINE(24);GOTOXY(3,24);READ(CARDATA);
  LN:=LENGTH(CARDATA);
  WHILE LN<45 DO
    BEGIN CARDATA:=CARDATA+' ';LN:=LN+1;END;
END;

PROCEDURE SEARCHBYLICENCENO;
LABEL E1;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('SEARCH BY LICENCE-NUMBER PROCEDURE ');
  ACCEPTLICENCENO;
  CALCINDXRECNO;GETINDXRECORD;SEARCHINDXRECORD;
  IF INDXPOSN=0 THEN
    BEGIN
    GOTOXY(3,22);WRITE('NOT IN THE MASTER FILE');
    GOTO E1;
    END;
  CONVERTSTRING(INDEXBUFF.POINTER[INDXPOSN],RECNO);
  GETMASTERRECORD;
  DISPLAYMASTERRECORD;
E1:
  WAITTOEXIT;
END;

PROCEDURE ADDNEWRECORD;
LABEL E1;
VAR I:INTEGER;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('ADD NEW RECORD PROCEDURE ');
  GETFIRSTMASTERRECORD;
  IF NEXTAVAILRECORD>=MAXMASTER THEN
    BEGIN
    GOTOXY(3,22);WRITE('MASTER FILE IS FULL, CANNOT UPDATE');
    GOTO E1;
    END;
  ACCEPTLICENCENO;
  CALCINDXRECNO;GETINDXRECORD;SEARCHINDXRECORD;
  IF INDXPOSN<>0 THEN
    BEGIN
    GOTOXY(3,22);WRITE('DUPLICATE LICENCE, CANNOT UPDATE');
    GOTO E1;
    END;
  FINDEMPTYINDEX;
  IF (INDXPOSN=0) THEN
    BEGIN
    GOTOXY(3,22);WRITE('INDEX RECORD IS FULL, CANNOT UPDATE');
    GOTO E1;
    END;
  ACCEPTCARDATA;
  MASTERBUFF.STATUS:='1';
  MASTERBUFF.ID:=LICNO;
  MASTERBUFF.DATA:=CARDATA;
  RECNO:=NEXTAVAILRECORD;
  DISPLAYMASTERRECORD;
  PUTMASTERRECORD;
  INDEXBUFF.KEY[INDXPOSN]:=LICNO;
  CONVERTNUM(NEXTAVAILRECORD,INDEXBUFF.POINTER[INDXPOSN]);
  PUTINDXRECORD;
  PUTFIRSTMASTERRECORD;
  GOTOXY(3,22);WRITE('UPDATE COMPLETED');
E1:
  WAITTOEXIT;
END;

PROCEDURE CHANGEDATA;
LABEL E1;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('CHANGE DATA PROCEDURE ');
  ACCEPTLICENCENO;
  CALCINDXRECNO;GETINDXRECORD;SEARCHINDXRECORD;
  IF INDXPOSN=0 THEN
    BEGIN
    GOTOXY(3,22);WRITE('NOT IN THE MASTER FILE');
    GOTO E1;
    END;
  CONVERTSTRING(INDEXBUFF.POINTER[INDXPOSN],RECNO);
  GETMASTERRECORD;
  DISPLAYMASTERRECORD;
  ACCEPTCARDATA;
  MASTERBUFF.DATA:=CARDATA;
  DISPLAYMASTERRECORD;
  PUTMASTERRECORD;
  GOTOXY(3,22);WRITE('UPDATE COMPLETED');
E1:
  WAITTOEXIT;
END;

PROCEDURE DELETERECORD;
LABEL E1;
VAR CH:CHAR;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('DELETE RECORD PROCEDURE ');
  ACCEPTLICENCENO;
  CALCINDXRECNO;GETINDXRECORD;SEARCHINDXRECORD;
  IF INDXPOSN=0 THEN
    BEGIN
    GOTOXY(3,22);WRITE('NOT IN THE MASTER FILE');
    GOTO E1;
    END;
  CONVERTSTRING(INDEXBUFF.POINTER[INDXPOSN],RECNO);
  GETMASTERRECORD;
  DISPLAYMASTERRECORD;
  GOTOXY(3,23);WRITE('PLEASE CONFIRM (Y/N)');
  GOTOXY(3,24);REPEAT UNTIL KEYPRESSED;
  READ(KBD,CH);
  IF CH<>'Y' THEN
    BEGIN
    GOTOXY(3,22);WRITE('TRANSACTION CANCELED');
    GOTO E1;
    END;
  MASTERBUFF.STATUS:='2';
  PUTMASTERRECORD;
  INDEXBUFF.KEY[INDXPOSN]:='XXXXXXXX';
  INDEXBUFF.POINTER[INDXPOSN]:='000';
  PUTINDXRECORD;
  GOTOXY(3,22);WRITE('RECORD DELETED');
E1:
  WAITTOEXIT;
END;

PROCEDURE BROWSERECORD;
LABEL L1,L2,E1;
VAR CH:CHAR;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('BROWSING PROCEDURE ');
  RECNO:=0;
L1:
  GETMASTERRECORD;
  CLEARLINE(7);CLEARLINE(8);
  CLEARLINE(9);CLEARLINE(10);
  DISPLAYMASTERRECORD;
L2:
  GOTOXY(3,23);WRITE('1=FOREWARD, 2=BACKWARD, 9=END'
  GOTOXY(3,24);REPEAT UNTIL KEYPRESSED;
  READ(KBD,CH);
  IF CH='9' THEN GOTO E1;
  IF (CH='1') AND (RECNO<MAXMASTER) THEN
    BEGIN RECNO:=RECNO+1;GOTO L1;END;
  IF (CH='2') AND (RECNO>0) THEN
    BEGIN RECNO:=RECNO-1;GOTO L1;END;
  BEEP;BEEP;
  GOTO L2;
E1:
  CLEARLINE(23);CLEARLINE(24);
END;

PROCEDURE REBUILDINDEXFILE;
LABEL E1;
VAR REBUILDERRSWT,SUM,I1,I2:INTEGER;
VAR AR100:ARRAY[1..MAXINDXRECORD] OF INDEXRECORD;
BEGIN
  NEWSCREEN;
  GOTOXY(30,5);WRITE('REBUILD THE INDEX FILE');
  REBUILDERRSWT:=0;
  FOR I1:=1 TO MAXINDX DO
    BEGIN
    INDEXBUFF.KEY[I1]:='XXXXXXXX';
    INDEXBUFF.POINTER[I1]:='000';
    END;
  FOR I2:=1 TO MAXINDXRECORD DO
    AR100[I2]:=INDEXBUFF;
```

รูปที่ 2 (ต่อ)

```
  GETFIRSTMASTERRECORD;
  FOR RECNO:=1 TO NEXTAVAILRECORD-1 DO
    BEGIN
    GETMASTERRECORD;
    IF MASTERBUFF.STATUS='1' THEN
      BEGIN
      LICNO:=MASTERBUFF.ID;
      GOTOXY(3,21);WRITE('RECNO=',RECNO,' LICENCE NO=',LICNO);
      CALCINDXRECNO;
      INDEXBUFF:=AR100[INDXRECNO];
      FINDEMPTYINDEX;
      IF INDXPOSN=0 THEN
        BEGIN
        GOTOXY(3,22);WRITE('INDEX RECORD IS FULL');
        REBUILDERRSWT:=1;
        GOTO E1;
        END;
      CONVERTNUM(RECNO,INDEXBUFF.POINTER[INDXPOSN]);
      INDEXBUFF.KEY[INDXPOSN]:=LICNO;
      AR100[INDXRECNO]:=INDEXBUFF;
      END;
    END;
  FOR INDXRECNO:=1 TO MAXINDXRECORD DO
    BEGIN
    INDEXBUFF:=AR100[INDXRECNO];
    PUTINDXRECORD;
    END;
E1:
  GOTOXY(30,6);
  IF REBUILDERRSWT=0 THEN WRITE('---OK')
     ELSE WRITE('---ERROR FOUND');
  WAITTOEXIT;
END;

BEGIN
  CLRSCR;
  ASSIGN(INDEXFILE,'FT1D01.DAT');
  ASSIGN(MASTERFILE,'FT1D02.DAT');
  RESET(INDEXFILE);
  RESET(MASTERFILE);

MAINLOOP:
  NEWSCREEN;
  GOTOXY(30,5);WRITE('MAIN MENU');
  GOTOXY(30,7);WRITE('1 = SEARCH BY LICENCE-NUMBER');
  GOTOXY(30,9);WRITE('2 = ADD NEW RECORD');
  GOTOXY(30,11);WRITE('3 = CHANGE DATA');
  GOTOXY(30,13);WRITE('4 = DELETE RECORD');
  GOTOXY(30,15);WRITE('5 = BROWSING');
  GOTOXY(30,17);WRITE('8 = REBUILD INDEX FILE');
  GOTOXY(30,19);WRITE('9 = END JOB');
  GOTOXY(3,23);WRITE('YOUR SELECTION?');

READKBD:
  GOTOXY(3,24);
  REPEAT UNTIL KEYPRESSED;
  READ(KBD,CH);
  IF (CH='1') THEN BEGIN SEARCHBYLICENCENO;GOTO MAINLOOP;END;
  IF (CH='2') THEN BEGIN ADDNEWRECORD;GOTO MAINLOOP;END;
  IF (CH='3') THEN BEGIN CHANGEDATA;GOTO MAINLOOP;END;
  IF (CH='4') THEN BEGIN DELETERECORD;GOTO MAINLOOP;END;
  IF (CH='5') THEN BEGIN BROWSERECORD;GOTO MAINLOOP;END;
  IF (CH='8') THEN BEGIN REBUILDINDEXFILE;GOTO MAINLOOP;END;
  IF (CH='9') THEN GOTO EOJ;
  BEEP;BEEP;
  GOTO READKBD;

EOJ:
  CLOSE(INDEXFILE);
  CLOSE(MASTERFILE);
  CLEARLINE(23);CLEARLINE(24);
  GOTOXY(3,23);WRITE('END OF JOB');
  END.
```

■

ขึ้นใหม่ได้ เราควรทำเรื่องนี้เมื่อเกิดไฟฟ้าดับหรือเกิดการหยุดโปรแกรมโดยกะทันหัน

บทสรุป

ผมคิดว่าตัวอย่างที่ให้ไว้คงจะพอชี้ให้เห็นว่า การออกแบบโครงสร้างของแฟ้มข้อมูลให้ทนมือทนเท้านั้นไม่ยากเลย โปรแกรมเหล่านี้สั้นและง่ายต่อการเข้าใจ สำหรับผู้ที่จะนำไปใช้งานจริง ๆ นั้นก็คงพอจะตัดแต่งและเพิ่มเติมเองได้ สำหรับงานที่มีปริมาณข้อมูลมากกว่าที่ทำเป็นตัวอย่างเอาไว้ ท่านก็อาจขยายขนาดโดยเพิ่มค่าของ MAXINDX, MAXINDXRECORD และ MAXMASTER ซึ่งประกาศเอาไว้ใน CONST (บรรทัดที่ 3 ของโปรแกรมทั้ง 2) และเพิ่มจำนวนดิจิตที่จะใช้เก็บตัวชี้ (Pointer) ให้สอดคล้องกัน ไฟล์แบบนี้จะเข้าถึงข้อมูลโดยการ READ 2 ครั้งเท่านั้น มันจึงทำงานได้เร็วมาก